# Programación Evolutiva

## Algoritmo de Koza. Camino de Santa Fe

## Práctica 3

**Índice**

# 1. Introducción.

*El objetivo de esta práctica consiste en el diseño de una hormiga artificial capaz de encontrar toda la comida situada a lo largo de un rastro irregular. El objetivo es utilizar programación genética para obtener el programa que realiza esta tarea.*

En este caso, cada uno de los individuos de la población, esto es, cada uno de los cromosomas que intervienen en el proceso, está representado por un árbol. Cada cromosoma es un posible programa que la hormiga puede utilizar para buscar la comida, y se identifica con los nodos del árbol, que pueden ser:

1. Terminales (nodos hoja): avanzar, girar a la derecha, girar a la izquierda.
2. No terminales:
    a. Función SIC(a, b): ejecuta *a* si hay comida delante. Si no, ejecuta *b*.
    b. Función Progn2(a, b): encadena las acciones *a* y *b*.
    c. Función Progn3(a, b, c): encadena las acciones *a*, *b* y *c*.

Así pues, sobre cada cromosoma de cada población se llevarán a cabo los procedimientos típicos de la programación genética y del algoritmo de Koza:

- Selección: emplearemos los métodos de selección que habíamos desarrollado en prácticas anteriores (ruleta, torneo determinista, torneo probabilístico, ranking y un método propio).
- Cruce: queremos unir dos cromosomas para obtener otros dos nuevos en base a ellos. Para ello usaremos el intercambio de subárboles: se eligen al azar dos nodos de cada uno de los progenitores y se intercambian los subárboles que cuelgan de ellos.
- Mutación: para esta práctica usaremos los siguientes métodos de mutación:
    a. Mutación de terminal: seleccionamos un terminal al azar del árbol y lo sustituimos por otro diferente.
    b. Mutación de función: seleccionamos un nodo no terminal del árbol y lo cambiamos por otro no terminal que represente una función de la misma aridad que tenía el primero.
    c. Mutación de árbol: se selecciona al azar un subárbol del cromosoma y se regenera aleatoriamente con una profundidad igual o menor al reemplazado.

Adicionalmente, en esta práctica se incluyen dos técnicas, el llamado *método de Tarpeian* y *el método bien fundamentado* para limitar el tamaño de los árboles que contiene cada individuo, penalizando la aptitud de aquellos que sean de mayor tamaño. En nuestro caso hemos considerado como de mayor tamaño aquellos que tienen un mayor número de nodos.

Al igual que cualquier otra aplicación de la programación evolutiva, tras un número determinado de iteraciones del algoritmo (generaciones) nos quedaremos con el mejor programa (cromosoma) que hayamos podido obtener. Para evaluar la aptitud de cada programa obtenido se ejecutará cada uno de ellos un número de iteraciones prefijado, valorando la cantidad de comida que consigue la hormiga con cada uno de ellos.

# 2. Resultados más significativos.

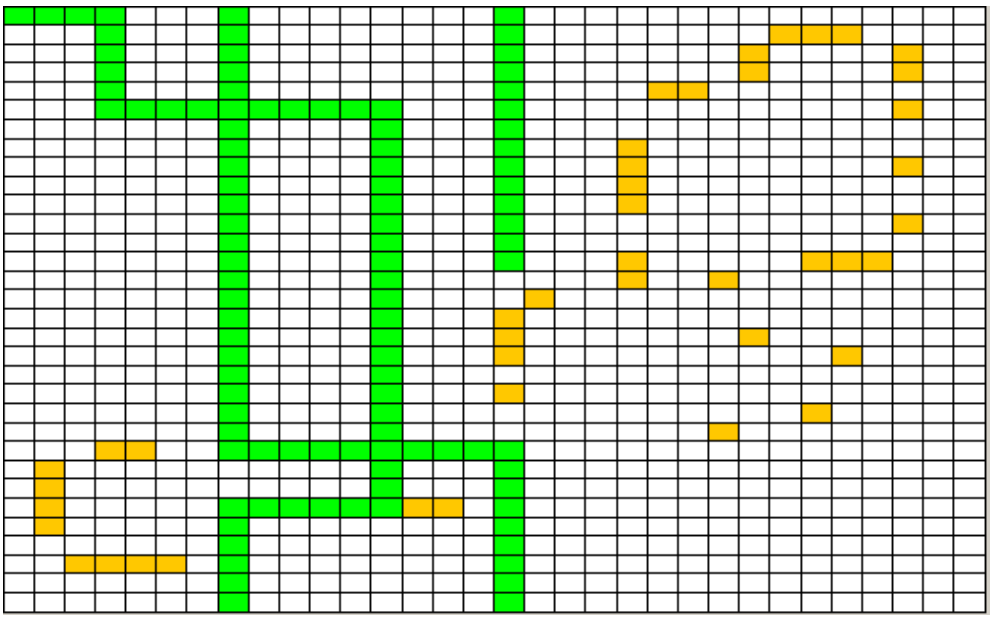
## 2.1. Ejecución habitual

Para el primer ensayo vamos a utilizar lo que hasta esta práctica hemos se ha considerado una parametrización tipo:

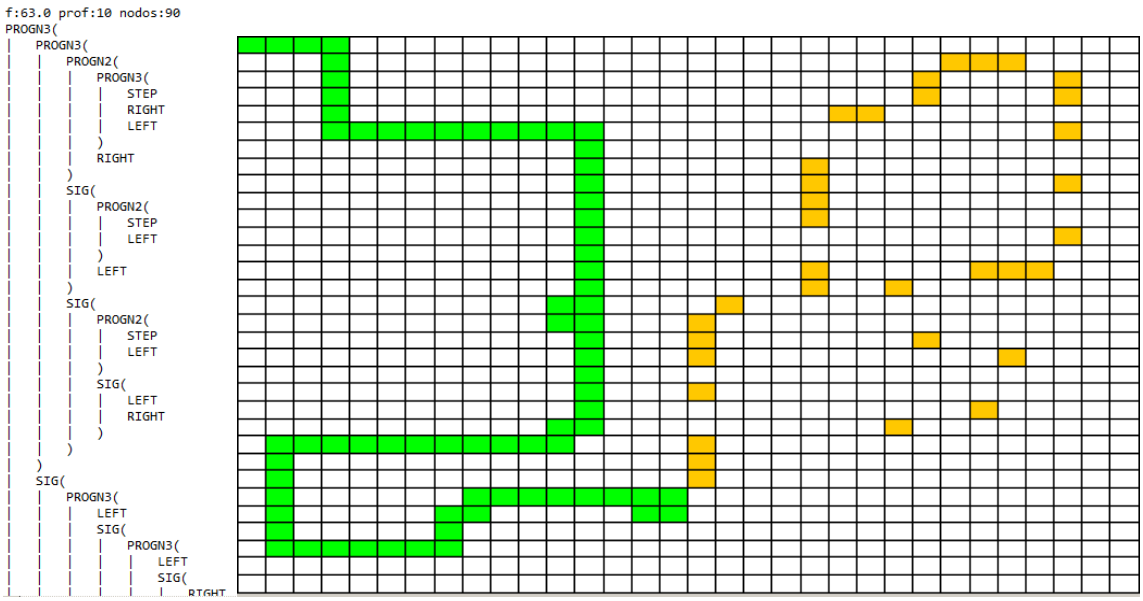
| T. Sel | T. Cruce | T. Mut | Bloating | Eli | T.Pob | N. Gen | PC | PM | Pasos | Prof. Min y Max |
|---|---|---|---|---|---|---|---|---|---|---|
| Ruleta | Intercambio de árboles | Terminal | No | No | 100 | 100 | 0.6 | 0.6 | 400 | (2,3) |

Se pueden obtener resultados aceptables en términos de la relación entre cantidad comida y tamaño del programa resultante como el siguiente caso:

```
f:47.0 prof:4 nodos:22
PROGN2(
|   PROGN2(
|   |   PROGN2(
|   |   |   SIG(
|   |   |   |   STEP
|   |   |   |   LEFT
|   |   |   )
|   |   |   RIGHT
|   |   )
|   |   PROGN2(
|   |   |   LEFT
|   |   |   LEFT
|   |   )
|   )
|   SIG(
|   |   SIG(
|   |   |   PROGN2(
|   |   |   |   RIGHT
|   |   |   |   LEFT
|   |   |   )
|   |   |   LEFT
|   |   )
|   |   PROGN2(
|   |   |   RIGHT
|   |   |   PROGN3(
|   |   |   |   STEP
|   |   |   |   STEP
|   |   |   |   RIGHT
|   |   |   )
|   |   )
|   )
`
```



Hasta resultados inaceptables con más de 70 nodos:




Para este tipo de parametrizaciones los resultados tienden a ser bastante aleatorios y suelen oscilar entre las 40 y 65 piezas de comida alcanzadas en 400 pasos, aunque se trata de árboles con una profundidad enorme (entre 4 y 10) y en consecuencia un número de nodos elevado.

### 2.2. Aumento del tamaño de la población

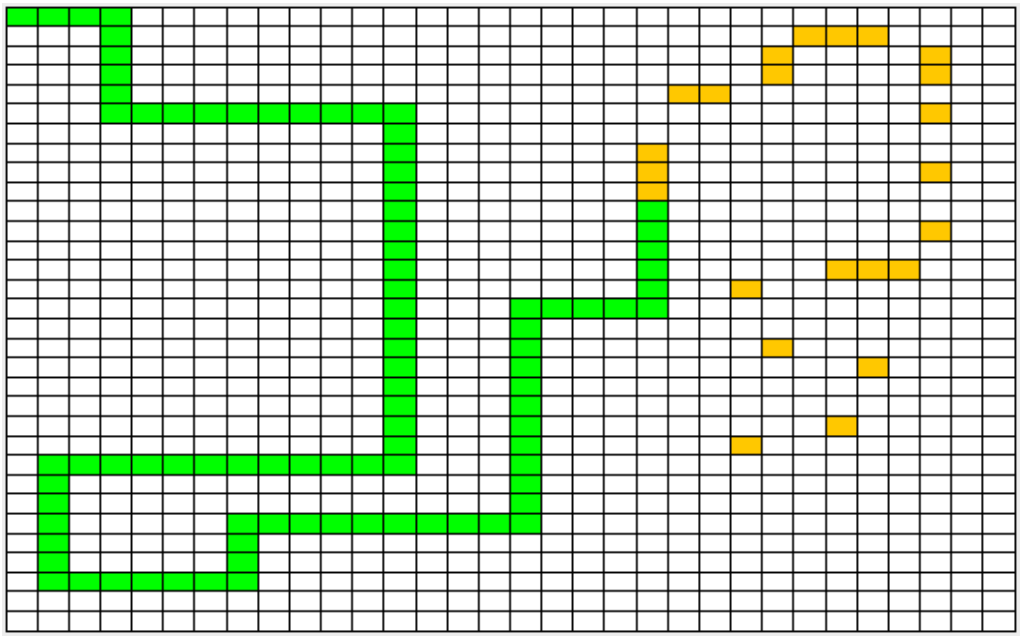
A priori, un aumento del tamaño de la población introducirá mayor diversidad en la generación inicial y en las sucesivas, permitiendo encontrar así mejores soluciones y mejorando el rendimiento de los algoritmos obtenidos.
Podemos observar que conforme aumentamos el tamaño de la población vemos que, por lo general, se obtienen mejores resultados que en el apartado anterior. Se aprecia un mayor rendimiento en una ejecución promedio, aunque lo cierto es que se siguen alternando resultados bastante buenos con algunos mediocres (tanto menos frecuentes cuanto mayor es el tamaño de la población, claro está).
Sin embargo, esta mejora no tiene lugar en términos del tamaño del programa ya que se sigue sin aplicar ninguna técnica de bloating.

Supongamos una ejecución de este tipo:

| **T. Sel** | **T. Cruce** | **T. Mut** | **Bloating** | **Eli** | **T.Pob** | **N. Gen** | **PC** | **PM** | **Pasos** | **Prof. Min y Max** |
|---|---|---|---|---|---|---|---|---|---|---|
| Ruleta | Intercambio de árboles | Terminal | No | No | [100-500] | 100 | 0.6 | 0.6 | 400 | (2,3) |

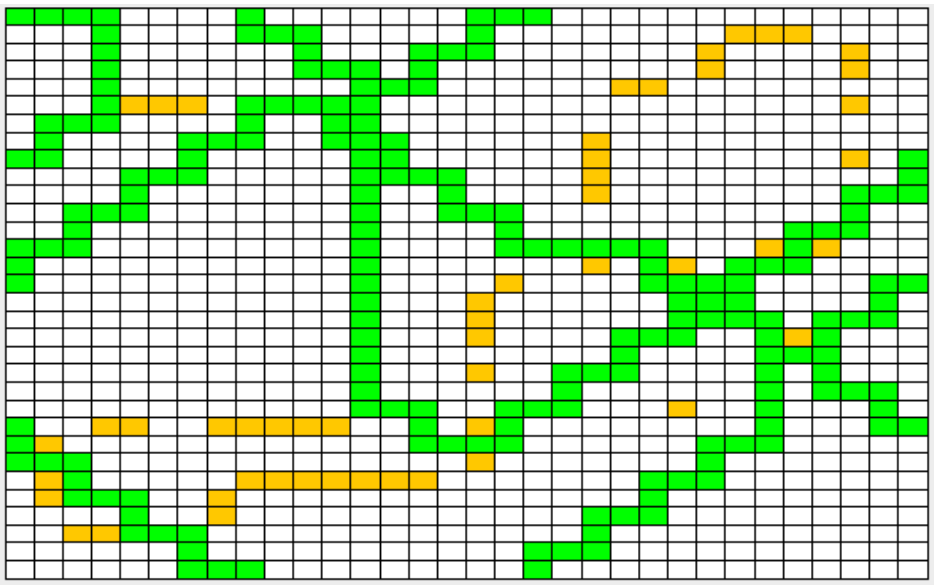
En el caso en el que la población es 200 obtenemos lo siguiente (entre otros resultados):

```
f:67.0 prof:3 nodos:15
PROGN3(
|   PROGN3(
|   |   SIG(
|   |   |   RIGHT
|   |   |   RIGHT
|   |   )
|   |   SIG(
|   |   |   STEP
|   |   |   LEFT
|   |   )
|   |   LEFT
|   )
|   SIG(
|   |   STEP
|   |   SIG(
|   |   |   STEP
|   |   |   RIGHT
|   |   )
|   )
|   STEP
)
```



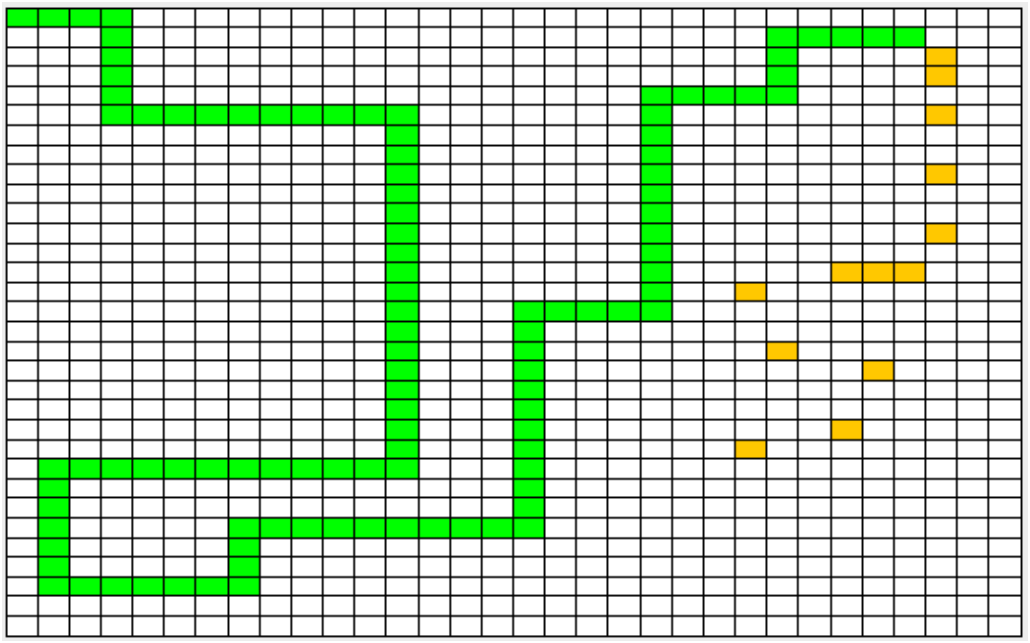
En este caso, vemos que hemos obtenido un resultado bastante aceptable con un programa relativamente corto y poco profundo. Esto nos llevaría a suponer que aumentando drásticamente el tamaño de la población con, por ejemplo, 500 individuos, conseguiríamos muy buenos resultados:

```
f:38.0 prof:3 nodos:20
PROGN3(
|   SIG(
|   |   PROGN2(RIGHT, LEFT)
|   |   RIGHT
|   )
|   SIG(
|   |   SIG(STEP, RIGHT)
|   |   PROGN3(STEP, STEP,
LEFT)
|   )
|   PROGN3(
|   |   LEFT
|   |   RIGHT
|   |   PROGN2(
|   |   |   STEP
|   |   |   STEP
|   |   )
|   )
)
```



Sin embargo, vemos que esto no siempre es así. La posibilidad de que mejorando las condiciones iniciales consigamos resultados menos buenos sigue exisitiendo. No obstante, lo cierto es que, por lo general, dicha mejora sí que se suele traducir en mejores resultados como este (para un tamaño de población de 500):

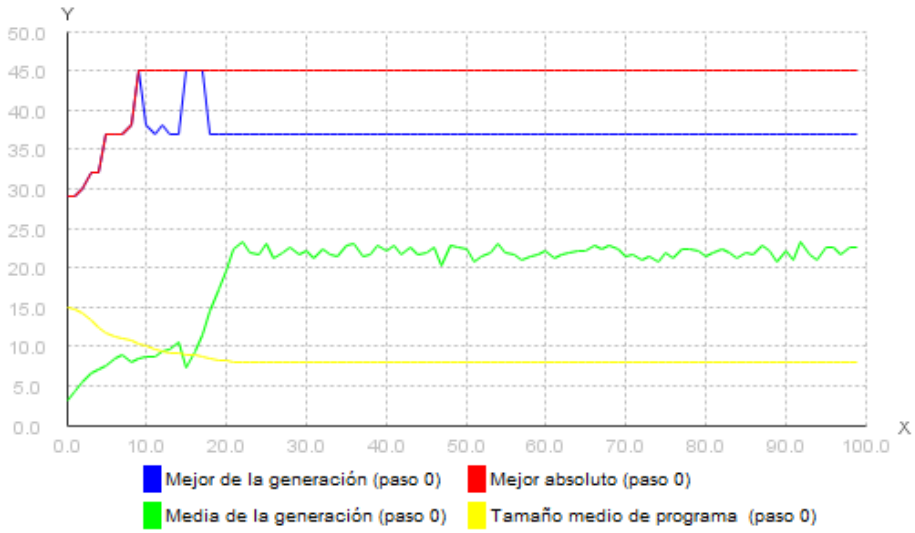
```
f:77.0 prof:3 nodos:13
PROGN3(
|   SIG(
|   |   PROGN2(
|   |   |   STEP
|   |   |   STEP
|   |   )
|   |   RIGHT
|   )
|   STEP
|   PROGN3(
|   |   RIGHT
|   |   SIG(
|   |   |   RIGHT
|   |   |   LEFT
|   |   )
|   |   LEFT
|   )
)
```



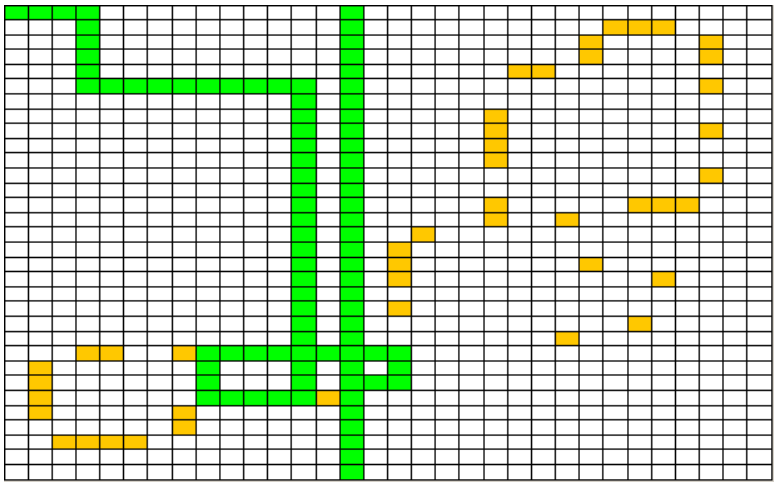
### 2.3. Ranking frente al resto de operadores de cruce

Al realizar las primeras pruebas utilizando parametrizaciones que en anteriores prácticas que habían ofrecido un buen resultado no parecían mostrar el comportamiento esperado en el algoritmo. Un ejemplo sería el siguiente resultado:

| T. Sel | T. Cruce | T. Mut | Bloating | Eli | T.Pob | N. Gen | PC | PM | Pasos | Prof. Min y Max |
|---|---|---|---|---|---|---|---|---|---|---|
| Ruleta | Intercambio de árboles | Terminal | Tarpeian | No | 300 | 100 | 0.6 | 0.6 | 400 | (2,3) |

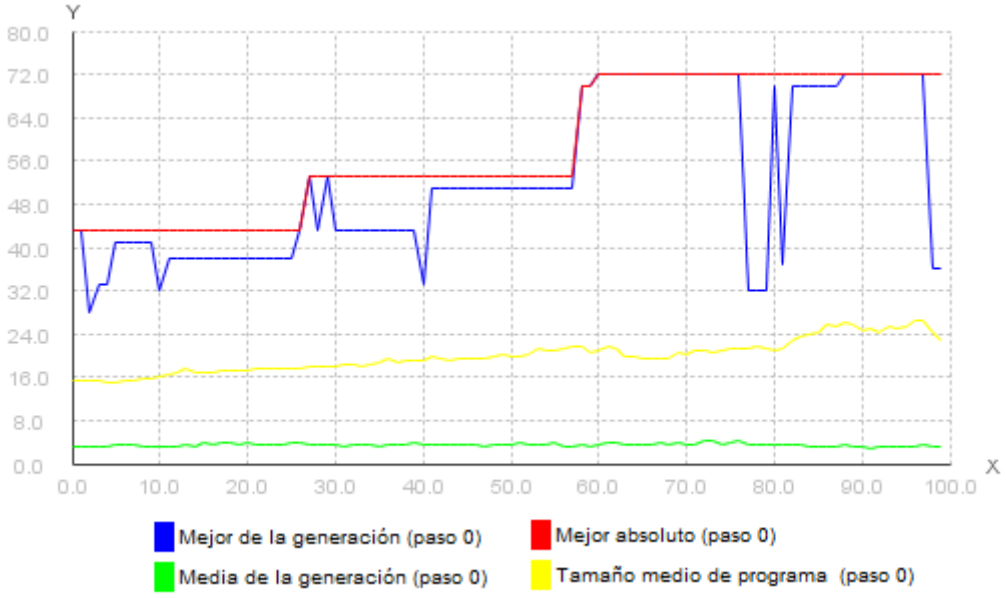


```
f:45.0 prof:3 nodos:9
PROGN2(
|   SIG(
|   |   STEP
|   |   LEFT
|   )
|   SIG(
|   |   STEP
|   |   PROGN2(
|   |   |   RIGHT
|   |   |   STEP
|   |   )
|   )
)
```

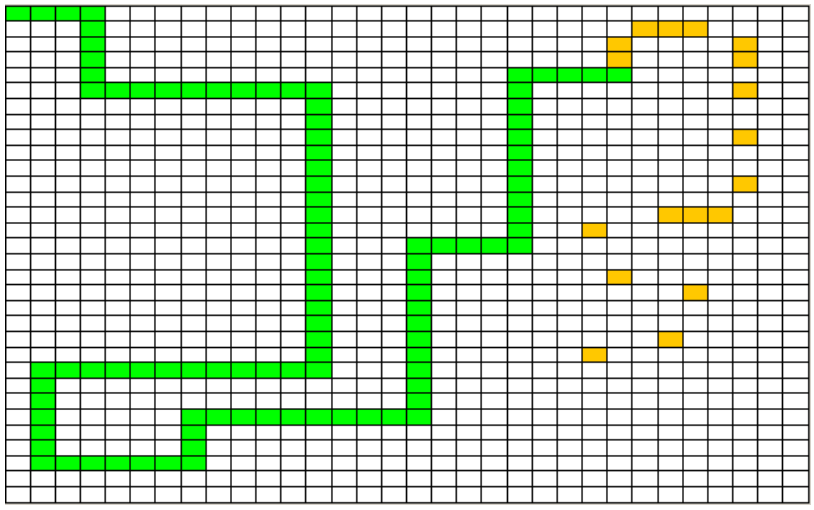


Resultados selección por ruleta

Por ello comenzamos a probar diferentes configuraciones en el proceso de selección y nos dimos cuenta que aquellas técnicas sobre las que tiene un menor peso la aptitud del individuo son las que ofrecían un mejor resultado. Esto se debe a que el efecto de las técnicas de control de bloating en menor en evoluciones con estos tipos de selección.
Las dos técnicas a las que nos referimos son la selección por ranking y la selección por torneo probabilístico. Aunque especialmente la primera para valores de beta próximos a su mínimo, es decir, uno.
Aquí tendríamos un ejemplo de dicha mejora:



```
f:72.0 prof:4 nodos:13
PROGN2(
|   PROGN2(
|   |   PROGN2(
|   |   |   STEP
|   |   |   LEFT
|   |   )
|   |   SIG(
|   |   |   STEP
|   |   |   PROGN2(
|   |   |   |   LEFT
|   |   |   |   LEFT
|   |   |   )
|   |   )
|   )
|   SIG(
|   |   STEP
|   |   LEFT
|   )
)
```



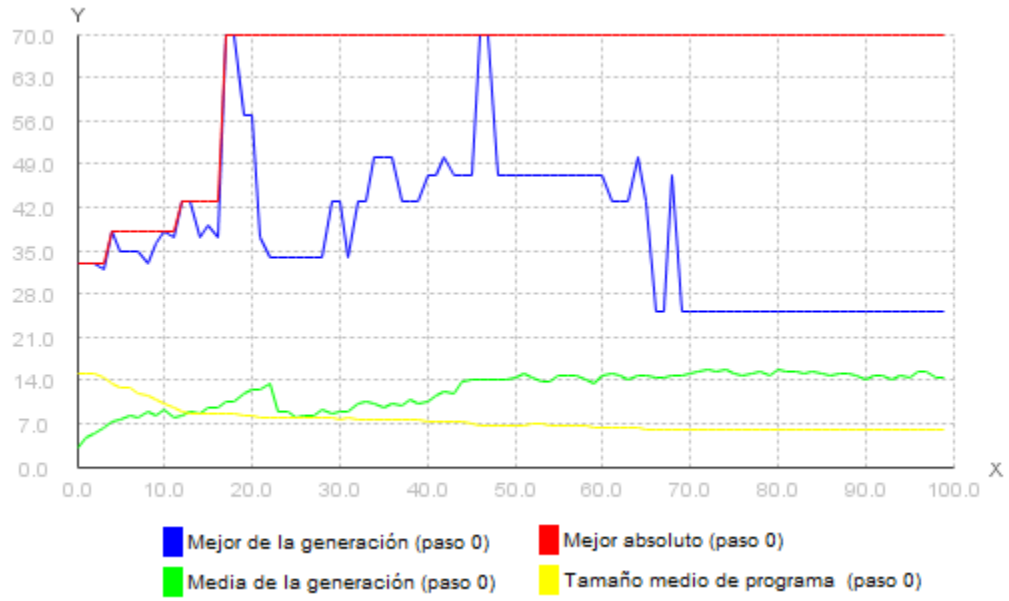
Resultados selección por ranking

### 2.4. Diferencias entre los métodos de mutación

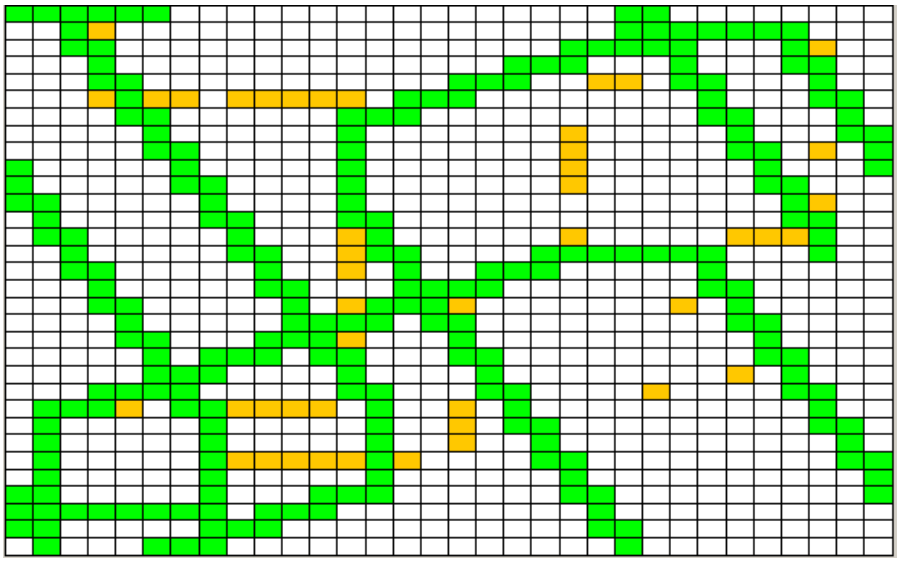
Para los siguientes ensayos consideraremos esta parametrización modificando en cada caso el método de mutación:

| T. Sel | T. Cruce | Bloating | Eli | T.Pob | N. Gen | PC | PM | Pasos | Prof. Min y Max |
|---|---|---|---|---|---|---|---|---|---|
| Ruleta | Intercambio de árboles | Tarpeian | No | 300 | 100 | 0.6 | 0.6 | 400 | (2,3) |

Los resultados en general mostrados tanto por la mutación de terminal simple como por la mutación funcional suelen ser bastante similares en todas las parametrizaciones.
Un posible resultado tipo para ambas sería el siguiente:

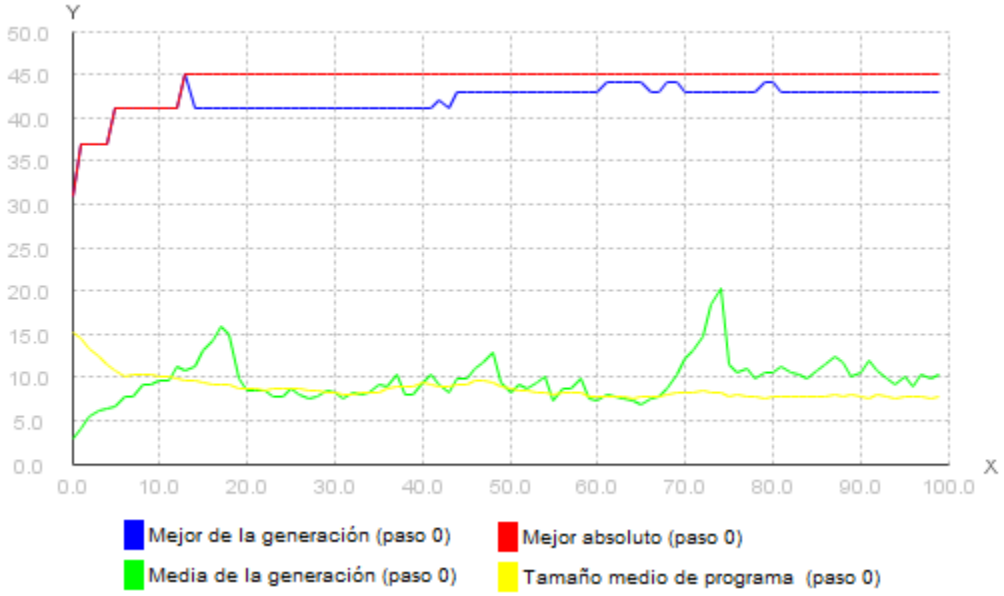


```
f:70.0 prof:4 nodos:13
PROGN3(
|   SIG(
|   |   STEP
|   |   LEFT
|   )
|   LEFT
|   SIG(
|   |   STEP
|   |   PROGN3(
|   |   |   SIG(
|   |   |   |   LEFT
|   |   |   |   RIGHT
|   |   |   )
|   |   |   STEP
|   |   |   RIGHT
|   |   )
|   )
)
```

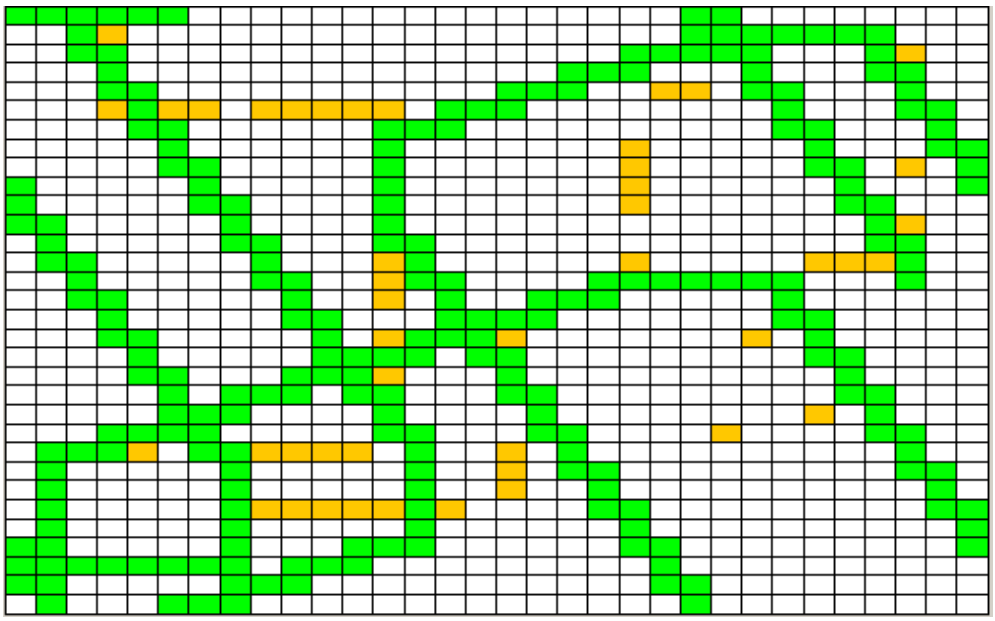


Resultados ejecución con mutación funcional

Pero aspecto más llamativo reside en la mutación de árbol. Este tipo de operación, aunque permita explorar nuevas alternativas en el espacio de búsqueda de forma más efectiva que las dos primeras puede tener un efecto negativo sobre la aptitud de los programas.
Al principio se consideró la posibilidad de no tener en cuenta la profundidad del árbol sustituido y generar árboles con una profundidad que fuese parametrizable en este tipo de mutación. El resultado fueron evoluciones con estrategias muy buenas pero terriblemente complejas. (En términos de profundidad y número de nodos)
Por ello se decidió sólo generar subárboles cuya profundidad fuese inferior o igual al número de nodos del árbol seleccionado para ser reemplazado. Aunque se logró que los tamaños de los resultados fueran similares a los de las otras técnicas,
Una posible ejecución para la misma parametrización que en las otras mutaciones sería la siguiente:



```
 f:45.0 prof:4 nodos:15
PROGN3(
|   SIG(
|   |   PROGN3(
|   |   |   PROGN3(
|   |   |   |   STEP
|   |   |   |   STEP
|   |   |   |   STEP
|   |   |   )
|   |   |   LEFT
|   |   |   RIGHT
|   |   )
|   |   RIGHT
|   )
|   PROGN3(
|   |   STEP
|   |   STEP
|   |   LEFT
|   )
|   STEP
)
```

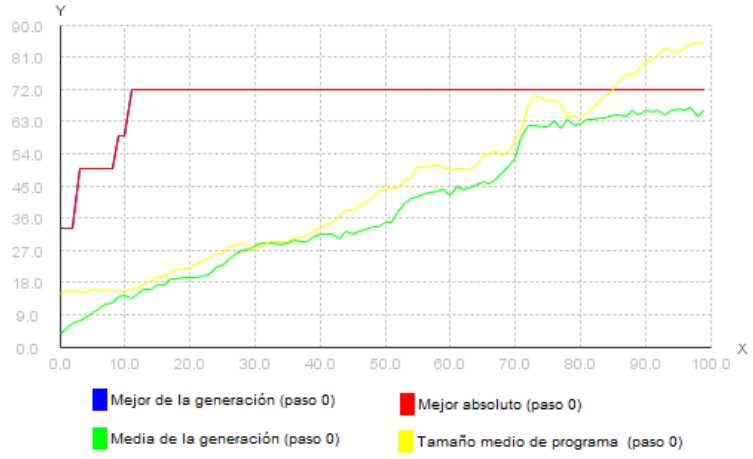


Resultados ejecución con mutación de árbol

### 2.5. Uso de técnicas de bloating

Como hemos visto en algunos de los ensayos anteriores, el tamaño del programa y la probabilidad de obtener una estrategia con excesivos movimientos redundantes sin especificar ninguna técnica de control de bloating se incrementa notablemente. Veamos por ejemplo la siguiente ejecución:

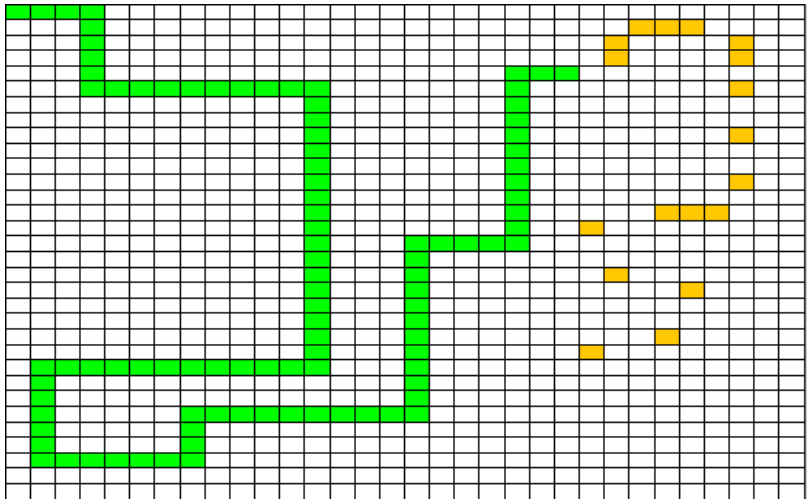
| T. Sel | T. Cruce | T. Mut | Bloating | Eli | T.Pob | N. Gen | PC | PM | Pasos | Prof. Min y Max |
|---|---|---|---|---|---|---|---|---|---|---|
| Ranking | Intercambio de árboles | Terminal | No | No | 300 | 100 | 0.6 | 0.6 | 400 | (2,3) |




Ejecución sin método control de bloating

Aunque como en este caso pueden obtenerse como resultado estrategias bastante efectivas:

```
f:72.0 prof:3 nodos:24
PROGN3(
|   PROGN3(
|   |   SIG(
|   |   |   LEFT
|   |   |   RIGHT
|   |   )
|   |   RIGHT
|   |   SIG(
|   |   |   STEP
|   |   |   LEFT
|   |   )
|   )
|   SIG(
|   |   PROGN2(
|   |   |   STEP
|   |   |   RIGHT
|   |   )
|   |   SIG(
|   |   |   STEP
|   |   |   RIGHT
|   |   )
|   )
|   SIG(
|   |   SIG(
|   |   |   STEP
|   |   |   RIGHT
|   |   )
|   |   PROGN3(
|   |   |   LEFT
|   |   |   STEP
|   |   |   LEFT
|   |   )
|   )
)
```

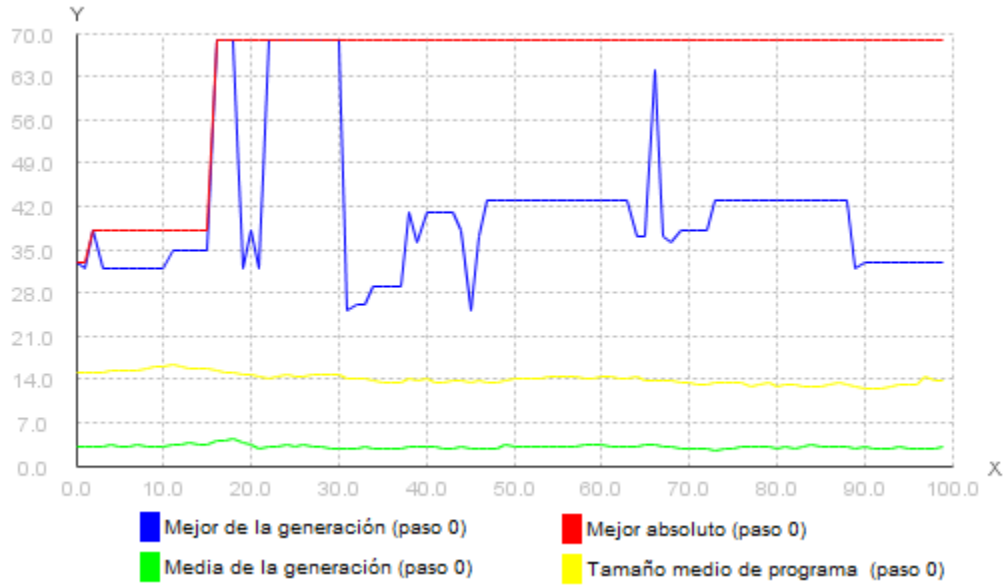


Resultados ejecución sin método control de bloating

Para poder controlar este fenómeno el número de nodos de los árboles, o lo que es lo mismo el tamaño de cada programa asociado a cada individuo, se han implementado varias técnicas de bloating.

La primera de ellas es el llamado *método Tarpeian* que penaliza la aptitud de los individuos con mayor tamaño que la media de la población. Podemos ver una ejecución de este método para la siguiente parametrización:

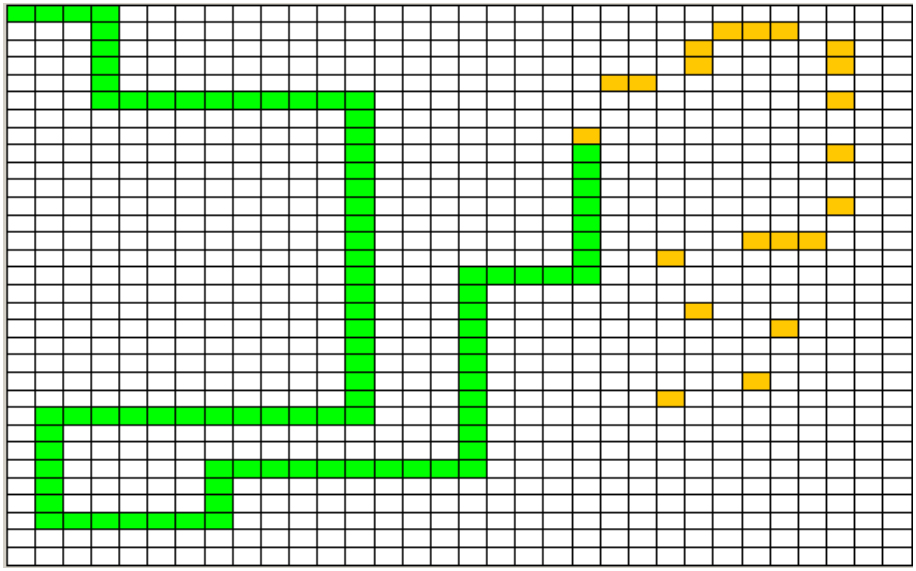
| **T. Sel** | **T. Cruce** | **T. Mut** | **Bloating** | **Eli** | **T.Pob** | **N. Gen** | **PC** | **PM** | **Pasos** | **Prof. Min y Max** |
|---|---|---|---|---|---|---|---|---|---|---|
| Ranking | Intercambio de árboles | Terminal | Tarpeian | No | 300 | 100 | 0.6 | 0.6 | 400 | (2,3) |




Ejecución *método Tarpeian*

Vemos como el tamaño medio de programa se estabiliza y se mantiene constante entorno a los catorce nodos por árbol.
También se puede observar como la media por generación decrece, por un lado por la penalización que está efectuando el método de bloating sobre buena parte de los individuos y por otra debido a un comportamiento bastante particular de nuestra práctica.
Cuando se escoge Tarpeian en lugar el mecanismo de reemplazamiento en el cruce varía. En lugar de utilizar un reemplazamiento por inclusión se utiliza un reemplazamiento inmediato lo que puede hacer que en ocasiones se reemplacen progenitores con una aptitud mucho mejor que los descendientes que los sustituyen. Esta decisión está fundamentada en razones experimentales, básicamente aunque en media empeore, ofrece mejores resultados finales.

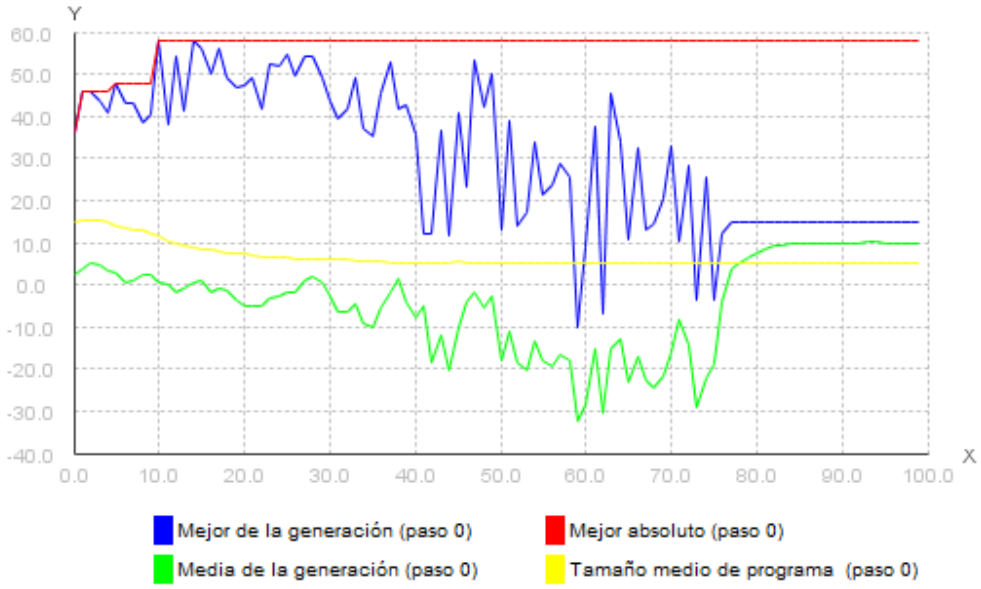
```
f:69.0 prof:3 nodos:16
PROGN2(
|   SIG(
|   |   STEP
|   |   SIG(
|   |   |   LEFT
|   |   |   LEFT
|   |   )
|   )
|   PROGN3(
|   |   PROGN2(
|   |   |   STEP
|   |   |   LEFT
|   |   )
|   |   SIG(
|   |   |   STEP
|   |   |   LEFT
|   |   )
|   |   SIG(
|   |   |   RIGHT
|   |   |   LEFT
|   |   )
|   )
)
```



Resultados ejecución *método Tarpeian*

Alternativamente hemos incluido otra técnica de control de bloating, el método Bien Fundamentado. La idea es mantener el tamaño medio de la población inicial a lo largo de todo el proceso evolutivo.
A continuación mostramos el resultado de una ejecución con este método:

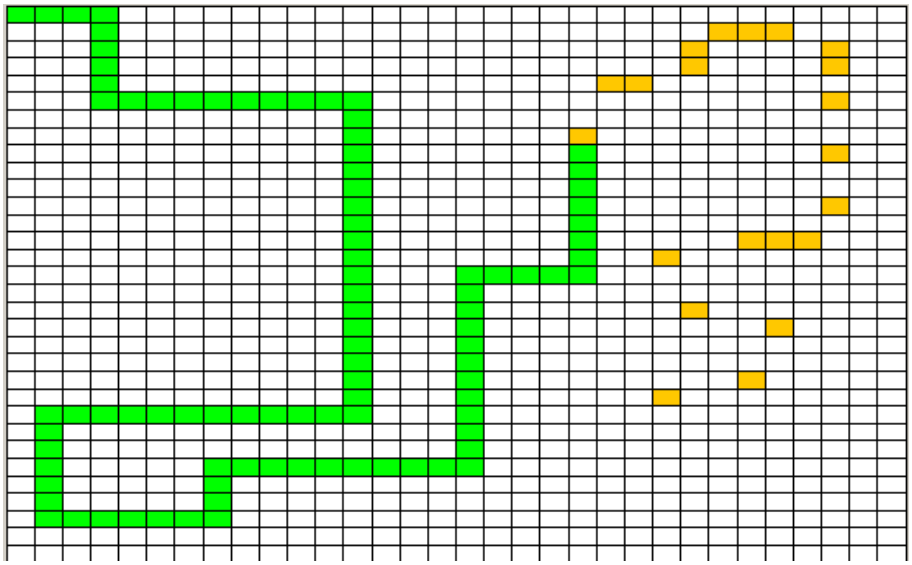
| T. Sel | T. Cruce | T. Mut | Bloating | Eli | T.Pob | N. Gen | PC | PM | Pasos | Prof. Min y Max |
|---|---|---|---|---|---|---|---|---|---|---|
| Ranking | Intercambio de árboles | Terminal | Bien fundamentado | No | 300 | 100 | 0.6 | 0.6 | 400 | (2,3) |




Ejecución *método bien fundamentado*

Como se puede observar el tamaño de programa se mantiene con valores similares a los de la primera generación. Aunque la técnica de bloating está repercutiendo negativamente en la media de la población

```
f:57 prof:4 nodos:16
PROGN2(
|   SIG(
|   |   SIG(
|   |   |   SIG(
|   |   |   |   LEFT
|   |   |   |   LEFT
|   |   |   )
|   |   |   STEP
|   |   )
|   |   RIGHT
|   )
|   PROGN3(
|   |   PROGN2(
|   |   |   RIGHT
|   |   |   SIG(
|   |   |   |   STEP
|   |   |   |   RIGHT
|   |   |   )
|   |   )
|   |   STEP
|   |   RIGHT
|   )
```



```
)
```

Resultados ejecución *método bien fundamentado*

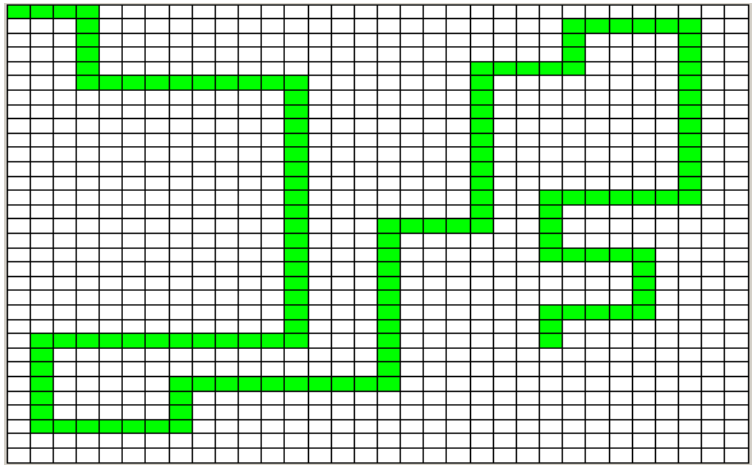
Pero como hemos visto en ambos casos, existe una mayor probabilidad de que el resultado final obtenido aplicando las técnicas de control de bloating sea mucho peor.
Hay que alcanzar un compromiso entre la cantidad de árboles penalizados por la técnica de control y el tamaño y calidad de los programas resultado.

### 2.6. Mejor resultado

Por último, sólo comentar alguna configuración a partir de la cual se puede obtener el mejor resultado posible, esto es, los noventa bocados.

| T. Sel | T. Cruce | T. Mut | Bloating | Eli | T.Pob | N. Gen | PC | PM | Pasos | Prof. Min y Max |
|---|---|---|---|---|---|---|---|---|---|---|
| Ranking | Intercambio de árboles | Terminal | Tarpeian | No | 600 | 100 | 0.6 | 0.6 | 600 | (2,3) |

```
f:90.0 prof:3 nodos:11
PROGN3(
|   SIG(
|   |   STEP
|   |   RIGHT
|   )
|   STEP
|   PROGN3(
|   |   RIGHT
|   |   SIG(
|   |   |   RIGHT
|   |   |   LEFT
|   |   )
|   |   LEFT
|   )
)
```



A medida que aumentamos el tamaño de población se obtienen resultados menos aleatorios y si además incrementamos el número máximo de pasos podemos obtener soluciones tan buenas como la anterior.

# 3. Conclusiones.

Al igual que en prácticas anteriores, se puede comprobar empíricamente que la mejor combinación de operadores y parámetros no es predecible, es necesario probar y variar cada uno de ellos para hallar los comportamientos que mejor rendimiento nos ofrezcan.
En nuestro caso, para este problema, hemos determinado que los mejores resultados, por lo general, los proporcionaba el método de selección por ranking combinado con la mutación de terminales o de función, con unos resultados sensiblemente peores con la mutación de árbol. En cuanto a los parámetros iniciales, es trivial decir que el aumento del tamaño de la población y el número de generaciones producen, por fuerza, mejores resultados. Sin embargo, un matiz no tan evidente, pero que se hace necesario enseguida, es que la probabilidad de mutación debe ser alta (más de 0,6 al menos): el algoritmo tiende a localizar y estancarse en máximos locales, estancamientos que se pueden resolver aumentando la influencia que la mutación tiene en el proceso.

En cuanto a la programación evolutiva, hemos podido observar que, al menos aplicando técnicas al nivel de las anteriores, resulta bastante más voluble. Si bien la evolución del proceso en prácticas anteriores era mayormente predecible a través de sus parámetros iniciales, esto no es así para esta. Múltiples ejecuciones con los mismos parámetros nos llevan a resultados de aptitud muy dispar, mientras que anteriormente las soluciones obtenidas por este método resultaban muy similares.